Lab 7 - Code Design II

Saturday, April 18, 2020 10:52 AM

Operation:
// Ambient Temperature controls operational mode

Initialize Continuous loop
- Read ambient temperature and convert to °F

```
 while(1) {
/* ------  read temp data ------      */

  RestartI2C1();                    // Send the stop condition
  IdleI2C1();                               // Wait to complete

  MasterWriteI2C1 ((SlaveAddress << 1) | 0x01);// transmit a read command
  IdleI2C1();                               // Wait to complete

  rd_cnt = MastergetsI2C1( 2, in_char_p, 152);   // read two bytes

  StopI2C1();                               // Stop I2C bus
  IdleI2C1();                               // wait to complete

 /* ---- calculate temp in degrees F -------- */

a = (short int) i2c_data[0];        // cast bytes to 16 bits
  b = (short int) i2c_data[1];
  raw_temp = b | (a << 8);                  // combine bytes to 9 bit result
  degc = raw_temp /256;                                     // shift out ls 7 bits of 0 - degrees C
  degf = ((degc * 9)/ 5) + 32;                      // Convert to Farenheit
```

- If ambient temperature is >= 80 °F | // Overheated
  - Flash message on LCD Display
  - Transmit message to terminal with CR and LF
  - Clear LEDs
  - Set overheat flag

```
if (degf >= TEMP_LIMIT) {

        // LCD module clear display command sequence
  SPI1BUF=0x1b;                  // Display clear - first send escape char
  c_buffer = "[j";               // command sequence for clear
  putsSPI1(2,c_buffer);          // write out string
  DelayMs(200);                                 // wait for display to clear

putsSPI1(12,buffer1);           // write out overheated string
  DelayMs(500);                  // wait - output will blink
```

```
SPI1BUF=0x1b;                   // Display clear - first send escape char
 c_buffer = "[j";               // command sequence for clear
 putsSPI1(2,c_buffer);      // write out string
 DelayMs(200);                              // wait for display to clear



    if (!temp_limit)  {
        temp_limit = 1;

        /*   write OVER Heated to UART2 - only once */
            WriteUART2(13 );        // carraige return - move cursor left
            putsUART2(buffer2);                 // write out Overheated string
        }
    led_bits = 0;                                       // set power to zero
    mPORTEWrite(led_bits);                              // write bits to output port
```

- Else
  - Read buttons and convert to level 0..7
  - Delay for switch bounce
  - Set output LEDs corresponding to power level
  - Display temperature on LCD Display
  - Display temperature on Terminal - stay on same line

```
else {
    temp_limit = 0;                 // reset flag in normal operation

    sprintf(str,"%d",degf );   // convert int to ASCII string

    /* ============ for now just dispaly temp ======== */

    SPI1BUF=0x1b;               // cursor reset - first send escape char
    c_buffer="[j";              // command sequence for clear screen
                                                        // and reset cursor
    putsSPI1(2,c_buffer);     // write out string

    DelayMs(100);               // wait for display to reset
    putsSPI1(strlen(str),str);// write out string
    DelayMs(100);               // wait for display to reset

    // Read buttons
    power = 0;                                  // determine power value each iteration
    button_in12 = PORTReadBits (IOPORT_G, BIT_6|BIT_7);
    button_in3 = PORTReadBits (IOPORT_A, BIT_0);

    if (button_in12 != 0){
        // drive both LD1 and LD2 high if both buttons pressed
        if (((button_in12 & 0x0040) != 0) && ((button_in12 & 0x0080) != 0))
            power = 3;
        else {
            //drive LD1 high if only BTN1 pressed
            if ((button_in12 & 0x0040) != 0)        // BTN1 pressed?
```

```
                power = 1;;
            //drive LD2 high if only BTN2 pressed
            if ((button_in12 & 0x0080) != 0) // BTN2 pressed
                power = 2;
            }
}
// Handle BTN3 separately
if(button_in3 !=0)
    power += 4;

/* power level determined - now illuminate LEDs */
switch (power) {
    case 0:
        led_bits = 0;
        break;
    case 1:
        led_bits = 0x1;
        break;
    case 2:
        led_bits = 0x3;
        break;
    case 3:
        led_bits = 0x7;
        break;
    case 4:
        led_bits = 0xf;
        break;
    case 5:
        led_bits = 0x1f;
        break;
    case 6:
        led_bits = 0x3f;
        break;
    case 7:
        led_bits = 0x7f;
        break;
    default:
        led_bits = 0;
        break;
}

mPORTEWrite(led_bits);                                          // write bits to output
port

WriteUART2(13 );    // carraige return - move cursor left
putsUART2(str);                                     // write out temp string
```